CIS 771: Software Specifications

Lecture 6: Dynamic Models in Alloy

Overview

- Implicit singleton sets
- Basics of dynamic models
  - States and transitions
  - Operation schema
  - Primed components
- Simple example operations
Indexed relations : Subtleties

- Alloy allows relational image to be lifted to sets
  
  \[
  \text{parent} = \{ \text{Matt} \rightarrow \text{Larry}, \text{Matt} \rightarrow \text{Valerie}, \text{Sean} \rightarrow \text{Matt}, \text{Kate} \rightarrow \text{Matt} \}
  \]

  If \( p = \{ \text{Matt} \} \) then \( p.\text{parent} = \{ \text{Larry, Valerie} \} \)

  If \( p = \{ \text{Matt,Sean} \} \) then \( p.\text{parent} = \{ \text{Larry, Valerie, Matt} \} \)

- Indexing relations does NOT lift to sets
  
  \[
  \text{matches}[\text{Matt}] = \{ \text{John} \rightarrow \text{Lori} \}, \text{matches}[\text{Sean}] = \{ \text{Bob} \rightarrow \text{Patty} \}
  \]

  If \( p = \{ \text{Matt}, \text{q = \{ John,Patty \} then q.matches[p] = \{ Lori \} \}

  If \( p = \{ \text{Matt, Sean} \}, q = \{ \text{John,Patty} \} \) then \( q.\text{matches}[p] = \text{ILLEGAL} \)

How can we be sure that we have a singleton set?
- Use a multiplicity
  \[
  P : \text{Person!} \ldots \text{some q.matches}[p]
  \]
- Use a predicate
  \[
  \text{one p \&\& some q.matches}[p]
  \]
- Quantification
  \[
  \text{all p,q | some q.matches}[p]
  \]

Note: \( p \) and \( q \) are implicitly constrained to be singleton sets

- Alloy is a bit restrictive with index expressions
  You can only use a quantified variable
So far we’ve used alloy to define the allowable values of state components:
- values of sets
- values of relations
An instance is a set of state component values that:
- Satisfies the constraints defined by multiplicities, invariants, selected conditions, ...

Person = {Matt, Sue}
Man = {Matt}
Woman = {Sue}
Single = {}
Married = {Matt, Sue}
wife = {Matt->Sue}
children = {}
siblings = {}

Person = {Matt, Sue, Sean}
Man = {Matt}
Woman = {Sue}
Single = {}
Married = {Matt, Sue}
wife = {Matt->Sue}
children = {Matt->Sean, Sue->Sean}
siblings = {Sue->Matt-Sean, Matt->Sue-Sean}
Dynamic Models

- Static models allow us to describe the legal states of the system
- We also want to be able to describe the legal transitions between states
  - A person is born before they can be married
  - A person gets married before having children
  - A person is a child until they die

Transitions

- A person is born
  - Extend domain
  - Add to gender partition (Man or Woman)
  - Add to Single partition
  - Modify other state components according to invariants, etc.
- NB: No requirement that a person has parents
Transitions

- Two people get married
  - Domain and gender partitions are unchanged
  - Marital status partitions (Man, Woman) change
  - Wife relation changes

- A person is born to parents
  - Extend domain
  - Add to gender partition (Man or Woman)
  - Add to Single partition
  - Modify children/parents relations
State Sequences

Person = {Sue}
Man = {}
Woman = Single = {Sue}
Married = {}
wife = children = {}
siblings = {}

Expressing Dynamism in Alloy

- Need a mechanism for denoting current and next states
- Need a mechanism for defining
  - When a transition can occur
  - What its effects are
  - What its parameters, if any, are
- Alloy does not support the description of sequences only individual transitions
  - If we show that all transitions are legal then all sequences will be legal
Denoting Next States

- Alloy uses the traditional logical notation
  - Z, OCL, JML, ..., all do the same
- **Priming** a state component refers to its name in the state after the transition
  - Person, Person'
  - Man, Man'
  - p.children, p.children'

Next States

Man = \{Matt\}

Man' = \{Matt\}

Single = \{Matt, Sue\}

Single' = {}

Married = {}

Married' = \{Matt, Sue\}

wife = {}

wife' = \{Matt->Sue\}
Next States

Changes to Person?

\[ \text{some sean : Person'} \mid \text{sean } \notin \text{Person} \]

Changes to Men?

\[ \text{Single'} = \text{Single} + \text{sean} \]

Changes to Married?

\[ \text{Married'} = \text{Married} \]

Changes to children?

\[ \text{father.children'} = \text{father.children} + \text{sean} \]

For you to do (pause here)

- Recall the academia example
- Write a formula that describes the effect of a student, s, enrolling in a course, c
- Write a formula that describes the effect of a student, s, dropping a course, c
- Write a formula that describes the effect of a student, s, forming a committee, consisting of his/her advisor and two other faculty members, f1 and f2
Alloy Operations

- A schema for defining transitions
- Syntactically its similar to a parameterized condition

\[
\text{op Name}(p_1 : S_1, p_2 : S_2, \ldots) \begin{cases} 
\text{“formula that references unprimed and primed state components”} \\
\end{cases}
\]

Operation’s Semantics

- Pre condition
  - Describes the states in which the operation can successfully perform its intended function
- Post condition
  - Describes the effects of the operation in generating the next state
- Frame condition
  - Describes what does not change between pre-state and post-state of a transition
- Design Contract
  - If a caller guarantees the pre-condition then they may rely on the post-condition holding
General Form of Operation

Distinguishing the pre, post and frame conditions provides useful documentation.

```
op Name(p1 : S1, p2 : S2, ...) {
    // Pre condition

    // Post condition

    // Frame condition
    // (special part of post-condition)
}
```

Example: Marriage

```
op Marriage(groom : Man!, bride : Woman!) {
    // Only single people can get married
    groom#bride in Single

    // Blood relatives cannot marry
    not BloodRelated(groom, bride)

    // After marriage bride is groom’s wife
    groom.wife’ = bride

    // After marriage bride and groom are married
    Married’ = Married+bride+groom
}
```
Operation Parameters

```
op Marriage(groom: Man!, bride: Woman!) {
    groom in Man && one groom
    bride in Woman && one bride

    groom+bride in Single
    not BloodRelated(groom, bride)

    groom.wife' = bride
    Married' = Married+bride+groom
}
```

For you to do

- Recall the academia example
- Write an operation that specifies a student adding a course
- Write an operation that specifies a student dropping a course
- Think about whether your specifications
  - state what changes
  - state what remains the same